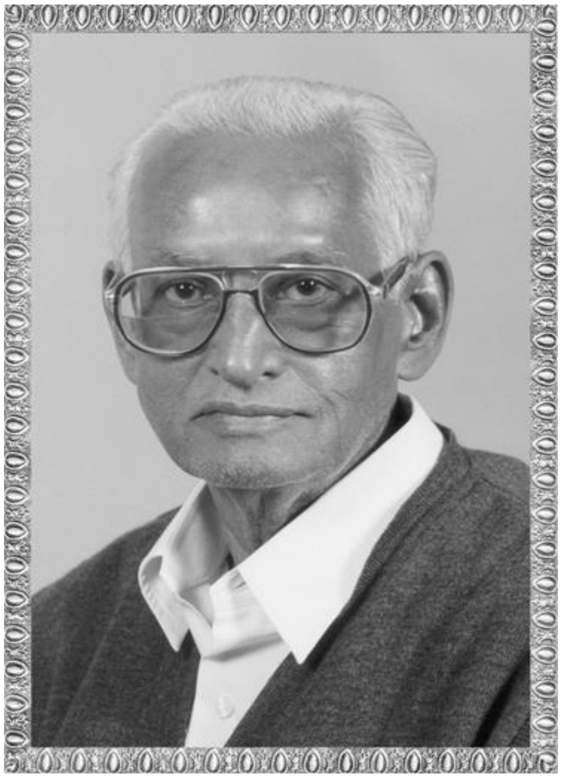
SHREE H. N. SHUKLA COLLEGE OF I.T & MGMT

(Affiliated to Saurashtra University)



Lt. Shree Chimanbhai Shukla

PGDCA – SEM-2 - C#.NET

Shree H.N.Shukla College,
Street No. 2, Vaishali Nagar,
Nr. Amrapali Railway Crossing,
Raiya Road, Rajkot.

Shree H.N.Shukla College,
Street No. 3, Vaishali Nagar,
Nr. Amrapali Railway Crossing,
Raiya Road, Rajkot.

Website:hnsgroupofcolleges.org
Email : hnsinfo@hnshukla.com

CS – 09: PROGRAMMING WITH C#		
Unit No.	**Topics**	**Details**
1	**Introduction**	Introduction to visual studio 2008 Visual studio editions Visual studio IDE
	C# Basics	▪ Variables, Constants, Strings ▪ Data types ▪ Arrays ▪ Decision statements ▪ Loop statements ▪ Exception using try-catch-finally ▪ NameSpace ▪ Class ▪ Object ▪ Struct
2	**Inheritance**	▪ Inheriting a class ▪ Sealed class ▪ Overloading an operator ▪ Overloading a method ▪ Overloading an Indexer ▪ Creating an Interface ▪ Implementing an Interface ▪ Inheriting an Interface
	Pointers and Delegates	▪ Pointers ▪ Pointers to Arrays ▪ Pointers to Structures ▪ Delegate ▪ Declaring and Instantiating Delegate ▪ Multicast delegate ▪ Creating events ▪ Chaining events ▪ Firing an event

3	**Threading in C#**	▪ Introduction ▪ Difference between process and thread ▪ The thread class ▪ Multithreading ▪ Thread Priorities ▪ Thread Synchronization
	Collection and Generics	**Understanding Collections:** ArrayList, BitArray, HashTable, Queue, SortedList, Stack, Generics, Generic List, Generic Stack, Generic Queue, GenericHashSet
4	**Reflection in C#**	Reflection, Why we need Reflection?, Using Reflection, Dynamic loading and reflection
	Windows Forms and Control Programming	Windows Forms: MsgBox, DialogBox, Handling Mouse, Events, Handling Key Events Basic Control Programming For Following: Controls, Button, Label, TextBox, RichTextBox, RadioButton, CheckBox ListBox, CheckedListBox, ComboBox, ListView, TreeView, ImageList, PictureBox Panel, GroupBox, TabControl, ScrollBar ToolTip, NotifyIcon, Timer, ProgressBar
5	**ADO.NET Programming**	Architecture of ADO. NET Data providers in ADO.NET: Connection Command DataReader DataAdapter DataSet: DataTable DataView DataColumn DataRow DataRelation DataReader DataGridView Control Introduction to LINQ Using LINQ to Dataset Example

Chapter-3 :- Threading

Topics	
Threading in C#	▪ Introduction ▪ Difference between process and thread ▪ The thread class ▪ Multithreading ▪ Thread Priorities ▪ Thread Synchronization
Collection and Generics	**Understanding Collections:** ArrayList, BitArray, HashTable, Queue, SortedList, Stack, Generics, Generic List, Generic Stack, Generic Queue, Generic HashSet

Topic: What is Thread?

Ans:

- Threads is a dispatchable unit of execution to which an operating system allocates processor time.
- Each thread defines a unique flow of control. If your application involves complicated and time consuming operations, then it is often helpful to set different execution paths or threads, with each thread performing a particular job.
- Threads are lightweight processes (Means of achieving mulit tasking).
- One common example of use of thread is implementation of concurrent programming by modern operating systems. Use of threads saves wastage of CPU cycle and increase efficiency of an application.

Topic: Differentiate Process and Thread

Ans:

Comparison Basis	Process	Thread
Definition	**It is a program under execution that is an active program**	**It is a lightweight process that can be managed independently by a scheduler.**
Context Switching Time (Time between 2 process)	**It requires more time**	**It requires less time**
Blocked	**If a process gets blocked, remaining processes can continue execution**	**If a user level thread gets blocked, all of its peer threads also gets blocked**
Resource Consumption	**Requires more resources**	**Requires less resources**

Dependency	**Individual processes are independent of each other**	**Threads are part of process and so they are dependent**
Data	**Processes have independent data**	**A thread shares data segment with its peer threads.**
Time for termination	**Process requires more time for termination**	**Thread requires less time for termination**

Topic: Explain Thread Class

Ans:

- .Net framework defines two types of thread: Foreground thread and background thread.
- When you create a thread, by default it is foreground thread, but you can change it to background thread.
- The only difference between foreground and background thread is that a background thread will be automatically terminated when all foreground threads in its process have stopped.
- Along with thread-based multitasking comes the need for a special type of feature called synchronization, which allows the execution of threads to be coordinated in certain well-defined ways
- The class that supports multithreading programming are defined in the System.Threading namespace.
- Thread class is sealed, which means that it can not be inherited.
- To create the thread, you have to create the object of thread class
    - Thread t=new Thread();
    - T.start();

- In the above syntax, start method is used to begin the execution of the thread.
- Once created, new thread will not start until you call its start method.
- If you try to call start() on a thread that has already been started, a ThreadStateException will be thrown.

Example:

```
class test
   {
      public void display()
      {
         Console.WriteLine("Hello");
      }
   }
   class Program
   {
      static void Main(string[] args)
      {
         test t1=new test();
         Thread t = new Thread(t1.display);
         t.Start();
         Console.ReadKey();
      }
   }
```

Topic: Explain States of Thread

Ans:

- States of thread can be checked using ThreadState enumerated property of the Thread object which contains different value for different states.
- Following are the states of Thread.

1) Unstarted: When the object of Thread class is created, it is in the unstarted state, means the thread has not yet started to run when the thread is in this state. Or in other words, start() is not called.

   Thread t=new Thread()

2) **Runnable State:** A thread that is ready to run is moved to runnable state. In this state, a thread might actually be running or it might be ready to run at any instant of time. It is the responsibility of the thread scheduler to give the thread, time to run. Or in other words, the *Start()* method is called.

3) **Running State:** A thread that is running. Or in other words, the thread gets the processor.

4) **Not Runnable State:** A thread that is not executable because
   a. Sleep() method is called.
   b. Wait() method is called.
   c. Due to I/O request.
   d. Suspend() method is called.
5) **Dead State:** When the thread completes its task, then thread enters into dead, terminates, abort state.

Example:

```
class test
    {
        public void display()
        {
            Console.WriteLine("Hello");
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            test t1 = new test();
            Thread t = new Thread(new ThreadStart(t1.display));
            Console.WriteLine(t.ThreadState);

            t.Start();
            Console.WriteLine(t.ThreadState);

            t.Suspend();
            Console.WriteLine(t.ThreadState);

            t.Resume();
            Console.WriteLine(t.ThreadState);

            Console.ReadKey();

        }
    }
```

Topic: Write a short note on Multithreading

Ans:

- Multithreading means when your program has more than one threads.
- **Multithreading in C#** is a process in which **multiple threads** work simultaneously.It is a process to achieve multitasking. It saves time because multiple tasks are being executed at a time.

Example:

```
class test
    {
        public void display()
        {
            Console.WriteLine("Hello");
        }
        public void display1()
        {
            Console.WriteLine("Hi");
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            test t1 = new test();
            Thread t = new Thread(new ThreadStart(t1.display));
            Thread t2 = new Thread(new ThreadStart(t1.display1));

             t.Start();
             t2.Start();
          Console.ReadKey();
        }
    }
```

Topic: Write a short note on Thread Priority

Ans:

- Each and every thread has its own priority settings
- Priority of thread means that how often thread gets access to CPU.
- Generally, high priority threads get more access to the CPU in compare to low priority threads.
- Thread can be assigned priority by using ThreadPriority
    - Highest
    - Lowest
    - Normal
    - BelowNormal
    - AboveNormal

Note: The default priority of thread is **ThreadPriority.Normal**

Example:

```
class test
    {
        public void display()//t1
        {
            Console.WriteLine("Hello");
        }
        public void display1()//t2
        {
            Console.WriteLine("Hi");
        }
        public void display2()
        {
            Console.WriteLine("how are you");
        }
        public void display3()
        {
            Console.WriteLine("aaaaa");
        }
        public void display4()
        {
            Console.WriteLine("bbbb");
        }
    }
```

```
class Program
   {
      static void Main(string[] args)
      {
         test t = new test();
     Thread t1 = new Thread(new ThreadStart(t.display));//hello
    Thread t2 = new Thread(new ThreadStart(t.display1));//hi
     Thread t3 = new Thread(new ThreadStart(t.display2));
     Thread t4 = new Thread(new ThreadStart(t.display3));
     Thread t5 = new Thread(new ThreadStart(t.display4));
     t1.Priority = ThreadPriority.Lowest;
     t3.Priority = ThreadPriority.Highest;
     t2.Priority = ThreadPriority.Normal;
     t4.Priority = ThreadPriority.BelowNormal;
     t5.Priority = ThreadPriority.AboveNormal;


    t1.Start();
   t2.Start();
   t3.Start();
   t4.Start();
   t5.Start();
     Console.ReadKey();
      }
   }
```

Topic: Write a short note on Thread Synchronization

Ans:

- The main disadvantage of multithreading is that we have to coordinate resources like files, network etc. Otherwise two or threads could not access same resource at same time.
- The process by which this is achieved is known as Synchronization.
- The most common reason for using synchronization is when two or more threads need access to shared resource that can be used by only one thread at a time.
- The key to synchronization is the concept of lock, which controls access to the block of code within an object. When the object is locked by one thread, another thread cannot gain the access to the locked block.
- When the thread releases the lock, the object is available for use by another thread.

Syntax:

```
lock(this)
{
        //statements;
}
```

Example:

```
class print
   {
      public void display()
      {
         lock (this)
         {

            for (int i = 0; i <= 10; i++)
            {
               Console.WriteLine(i);
            }
         }
      }
   }
   class Program
   {
      static void Main(string[] args)
      {
         print p = new print();
         Thread t = new Thread(new ThreadStart(p.display));
         Thread t1 = new Thread(new ThreadStart(p.display));
         t.Start();
         t1.Start();
         Console.ReadKey();
      }
   }
```

Topic: Write a short note on Collections in C#

Ans:

- **Collection represents group of objects.**
- Collection classes are specialized classes for data storage and retrieval. These classes provide support for stacks, queues, lists, and hash tables. Most collection classes implement the same interfaces.
- Collection classes serve various purposes, such as allocating memory dynamically to elements and accessing a list of items on the basis of an index etc.
- Namespace: System. Collections

Types of collection:

Non-Generic Collections:

- This type of collection store elements internally in object arrays so it can store any type of data.
- Following are the types of Non-Generic Collections:
  - ArrayList
  - Hashtable
  - Sorted List
  - Stack
  - Queue
  - BitArray

1) Array List

- Array List class is a collection that can be used for any types or objects.
- Arraylist is a class that is similar to an array, but it can be used to store values of various types.
- An Arraylist doesn't have a specific size.
- Any number of elements can be stored.
- Arraylist allocates memory for 4 items, whenever an object is created. When a fifth item is added, memory for another 4 items are added. It reduces the memory allocated for the object.

```
class Program
    {
        static void Main(string[] args)
        {
            ArrayList a = new ArrayList();
            string str = "hello how are u";
            int x = 10;
            char c = 'A';
            a.Add(str);
            a.Add(x);
            a.Add(c);

            foreach (object o in a)
                Console.WriteLine(o);
            Console.ReadKey();

        }
    }
```

Properties:

1) Capacity:
   It is the property that returns the number of items for which memory is allocated
2) Count:
   Gets the number of elements actually contained in arraylist
3) Contains:
   This property will return true if the element is present in the list, else it will return false.

Difference between Array and Array List	
Array	**ArrayList**
1) Array stores fixed number of elements	1) Arraylist can store any number of elements.
2) Array stores the elements of same data type	2) Arraylist stores the elements of different data types
3) Namespace: System.Array	3) Namespace: System.Collections
4) Arrays can not accept Null	4) ArrayList can accept Null
5) Array can be multidimensional	5)Arraylist is always single dimensional
6) We can add element in array by = operator	6) We can add elements in the arraylist by using Add().

MCQ	
1) Which property is used to count total number of elements in array?	**Count**
2) …..property returns the number of items for which memory is allocated	**Capacity**
3) ……method is used to add elements in the arraylist?	**Add()**

2) Hashtable:

- HashTable is similar to arraylist but represents the items as a combination of a key and value.
- Elements in the hashtable are stored as Dictionary Entry Objects.

```
class Program
   {
      static void Main(string[] args)
      {
         Hashtable t = new Hashtable();
         t.Add("1", "C#.Net");
         t.Add("2", "Asp.Net");
         t.Add("3", "J#.NEt");


         foreach (DictionaryEntry d in t)
            Console.WriteLine(d.Key + "" + d.Value);
         Console.ReadKey();
      }
   }
```

Properties:

1) Count: Gets the number of key and value pairs contained in hast table
2) Item: Gets or set the value associated with the specified key

Methods:

1) Add()): This method is used to add the element with specified key and value into hastable
2) Clear(): This method is used to remove all the elements from hashtable
3) ContainsKey(): This method determines whether the hashtable contains a specific key
4) ContainsValue(): This method determines whether the hash table contains a specific value or not
5) Remove(): This method is used to remove the element with specified key from the hashtable

Difference between ArrayList and Hashtable

ArrayList	Hashtable
1) Arraylist is based on index	1) Hash table is based on key
2) In array list, data is stored in form of index	2) In Hash table, data is stored in form of key and value
3) Dictionary Entry object is not used	3) Key value data is stored in Dictionary Entry object
4) Arraylist produces sequential output	4) Hash table does not produce sequential output.

MCQ	
1) …….method is used to clear all the elements of hash table	Clear()
2) ……..method is used to remove the element of particular key	Remove()
3) In hash table, data is stored in …….form	Key,value

3) Sorted List:

- It is the class that has the combination of arraylist and hastable
- Represents the data as key and value pair
- Arranges all the items in sorted order
- It contains a list of items that can be accessed using a key or an index. If you access items using an index, it is an ArrayList, and if you access items using a key, it is a Hashtable.

```
class Program
    {
        static void Main(string[] args)
        {
            SortedList s = new SortedList();

            s.Add("C#", "C#.Net");
            s.Add("Asp", "Asp.Net");
            s.Add("J#", "J#.NEt");
            foreach (DictionaryEntry d in s)
                Console.WriteLine(d.Key + "" + d.Value);
            Console.ReadKey();
        }
    }
```

Properties:

1) Capacity: Gets or sets the capacity of sorted list
2) Count: Gets the number of key and value pairs contained in sorted list
3) Item: Gets or set the value associated with the specified key in sorted list
4) Keys: Gets the key in sorted key
5) Value: Gets the value in sorted key

Methods:

1) Add()): This method is used to add the element with specified key and value into sorted list
2) Clear(): This method is used to remove all the elements from sorted list
3) ContainsKey(): This method determines whether the sorted list contains a specific key
4) ContainsValue(): This method determines whether the sorted list contains a specific value or not
5) GetByIndex(): Gets the key at specified index of the sorted list
6) IndexOfKey(): Returns the index of specified key in the sorted list
7) IndexOfValue(): Returns the index of first occurrence of the specified value in the given list
8) Remove(): This method is used to remove the element at specified key of sorted list
9) RemoveAt(): This method is used to remove the element at specified index of sorted list

MCQ	
1) ………is the combination of arraylist and hash table	**Sorted List**
2) Sorted list can access the data either through …….or …….	**Index, key**
3) Sorted List displays the output in	**True**

sorted form (T/F)	
4) ……….is used to remove the element at specified index	**RemoveAt()**
5) ………is used to remove the element at specified key	**Remove()**

4) Stack:

- A stack is a LIFO (Last In First Out) data structure
- Element in the stack are placed at the top and is removed from the top
- Example: Stack of plates or books
- To add element in stack, Push() is used and to remove the element from the stack, Pop() is used

```
class Program
   {
      static void Main(string[] args)
      {
         Stack s = new Stack();
         s.Push("Hello");
         s.Push("Hi");
         s.Push("xyz");

         foreach (object o in s)
            Console.WriteLine(o);
         s.Pop();
         foreach (object o in s)
            Console.WriteLine(o);
         Console.ReadKey();

      }
   }
```

MCQ	
1) Stack is based on …….	LIFO
2) To insert the elements in stack,…….method is used	Push()
3) To delete the elements in the stack, ………method is used	Pop()

5) Queue:

- A stack is a FIFO (First In First Out) data structure
- Example: Line of people
- To insert element in the queue Enqueue() is used and to remove the element from the queue Dequeue() is used.

```
class Program
  {
     static void Main(string[] args)
     {

        Queue q = new Queue();
        q.Enqueue("Hello");
        q.Enqueue("Hi");
        q.Enqueue("xyz");
        foreach (object o in q)
           Console.WriteLine(o);
        Console.ReadKey();
     }
  }
```

Difference between Stack and Queue	
Stack	**Queue**
1) Stack is based on LIFO	1) Queue is based on FIFO
2) In stack, insert operation is known as push	2) In Queue, insert operation is known as Enqueue
3) In stack, delete operation is known as pop	3) In Queue, delete operation is known as dequeue.

MCQ	
1) Queue is based on …….	**FIFO**
2) To insert the elements in queue,…….method is used	**Enqeue()**
3) To delete the elements in the queue, ………method is used	**Dequeue()**

6) Bit Array:

- The **BitArray** class stores array of bit values, which are represented as Booleans, where true indicates that the bit is **on** i.e., **1** and false indicates the bit is **off** i.e., **0**.

Example:

```
class Program
   {
       static void Main(string[] args)
       {

          BitArray a = new BitArray(3);
          a.Set(0, true);
          a.Set(1, false);
          a.Set(2, true);

          for (int i = 0; i < a.Count; i++)
             Console.WriteLine(a[i]);
          Console.ReadKey();
       }
}
```

Generic Collections:

- The Generic Collections store elements internally in arrays of their actual types.
- Namespace required for Generic collection is: using System.Collections.Generic
- A Generic Collection is strongly typed.
- Following are the types of Generic Collections
  - Generic Stack
  - Generic Queue
  - Generic List
  - Generic Hashset

1) Generic Stack:

- The **Generic Stack** in C# is a collection class which works on the principle of Last in First out (LIFO) and this class is present.
- **System.Collections.Generic** namespace.

```
class Program
   {
       static void Main(string[] args)
       {

           Stack<int> s = new Stack<int>();
           s.Push(10);
           s.Push(11);
           s.Push(12);
           s.Push(13);
           s.Push(14);

           foreach (int ele in s)
               Console.WriteLine(ele);
           Console.ReadKey();
       }
```

2) Generic Queue:

- The **Generic Stack** in C# is a collection class which works on the principle of Last in First out (LIFO) and this class is present.
- **System.Collections.Generic** namespace.

```
class Program
   {
      static void Main(string[] args)
      {

         Queue<int> q = new Queue<int>();
         q.Enqueue(10);
         q.Enqueue(11);
         q.Enqueue(12);
         q.Enqueue(13);
         q.Enqueue(14);

         foreach (int e in q)
            Console.WriteLine(e);
         Console.ReadKey();
      }
```

3) Generic List:

- **List<T> class** represents the list of objects which can be accessed by index.
- **System.Collection.Generic** namespace is used. List class can be used to create a collection of different types like integers, strings etc.

```
class Program
   {
      static void Main(string[] args)
      {
         List<int> f = new List<int>();
         f.Add(10);
         f.Add(11);
         f.Add(12);

         foreach (int e in f)
            Console.WriteLine(e);
         Console.ReadKey();
      }
   }
```